Rapport de projet de fin d'études

Composant d'audit des accès cache mémoire sur un softcore LEON3 en F.P.G.A.

Auteur :

Jérôme VAESSEN

Nom de la matière :

Projet de fin d'études

Année :

2014 / 2015

Encadrants-écoles :

Julien Iguchi-Cartigny

Pierrick Buret

Département Informatique Microélectronique et Automatique, 5ème année.

Remerciements

Je tiens tout d’abord à remercier, toute l’équipe pédagogique de Polytech’Lille et les responsables de la formation Informatique, Microélectronique et Automatique de m’avoir enseigné les bases pour réaliser ce projet.

Je tiens à remercier l'ensemble du personnel du laboratoire de l'Ircica, l’équipe 2XS, ainsi les deux encadrants qui m'ont accueilli au sein de l'équipe 2XS ; à savoir Pierrick Buret et Julien Iguchi-Cartigny.

Table des matières

Remerciements 2
Listes des figures 4
Listes des abréviations 5
Lexique 5
I. Présentation du problème 6
I.1. Contexte 6
I.2. Objectif du projet 6
I.3. Ressources utilisées 7
I.3.a. Carte utilisée 7
I.3.b. GRLIB de Gaisler 7
I.4. Cahier des charges du composant 8
I.4.a Position du composant dans le design 9
I.5. Étapes du projet 10
I.6. Gants prévisionnel 11
II. Implémentation du composant et du cache L2 12
II.2.Bus AMBA 12
II.3. Prise en main de l'interface de configuration de l'architecture avec la GRLIB 12
II.4. AHB 12
II.4.a. Maitres et Esclaves 12
II.4.b. PLUG & PLAY de l'arbitre de bus 13
II .4.c. Transfert AHB et allure des signaux d'accès d'un maître sur le contrôleur mémoire 14
II.5. Réalisation du composant 16
II.5.a. Schéma du composant 16
II.5.b. Test et correction 17
II.6. Réflexion sur un cache L2 18
II.6.a Etats de l'art : cache L1 18
II.6.b. Cache mémoire L2 19
II.6.c. Implantation et comportement 19
II.6.d Schéma du cache 19
III. Ressenti sur le projet 20
III.1. Difficultés rencontrées 20
III.2. Bilan personnel 21
IV. Diagramme de gant final 22
V. Conclusion 23
Annexes 24

Reconnaissance du composant 24
Lecture du compteur 24
ISE Xilinx workflow 24

Listes des figures

Figure 1 : Système LEON3 montrant le BUS AMBA AHB et APB 7
Figure 2 : Implantation dans le contrôleur mémoire 9
Figure 3 : Gant prévisionnel de septembre à décembre 11
Figure 4 : Gant prévisionnel de décembre à février 11
Figure 5 : Schéma de principe du contrôleur AHB (AHBCTRL) 12
Figure 6 : Interface de décodage selon la norme AMBA 2.0 13
Figure 7 : Simple transfert d'après la norme AMBA 14
Figure 8 : Extrait de la simulation du projet entier 15
Figure 9: Extrait des traces de simulation dans la console de ModelSim PE 15
Figure 10 : Composant simulé seul 16
Figure 11 : Schéma de principe du composant 16
Figure 12 : Schéma abstrait d'un processeur communiquant avec le contrôleur mémoire 18
Figure 13 : Chronogramme de deux tâches temps réels 18
Figure 14 : Schéma abstrait d'un processeur communiquant avec le contrôleur mémoire au travers du cache L2 19
Figure 15 : Proposition de schéma du cache pour l'entrée 20
Figure 17 : Gant réel de septembre à décembre 22
Figure 18: Gant réel de décembre à février 22
Figure 16 : Etapes de génération de "bitstream" 25
Figure 19 : Procedure de synthése 25

Listes des abréviations

FPGA : Field-Programmable Gate Array, réseau de portes programmables, c'est un circuit logique programmable

AMBA : Advanced Microcontroller Bus Architecture

AHB : Advanced High-performance Bus définit dans la norme AMBA

ASB : Advanced System Bus définit dans la norme AMBA

APB : Advanced High-performance Bus définit dans la norme AMBA

VHSIC : Very High Speed Integrated Circuit

VHDL : VHSIC Hardware Description Language

IP : Intellectual Properties

GRLIB : Gaisler Research LIBrary ; bibliothéques VHDL contenant des composants électroniques

GRMON : Gaisler Research MONitor

CPU : Central Processing Unit

JTAG : Joint Test Action Group est une liaison par câble définit par une norme IEEE

GNU GPL : GNU General Public License

ESA : Europeaon Space Agency

TCL/TK : langages, Tool Command Language Tk

PCI : Peripheral Component Interconnect

Lexique

Réseau CAN : (Controller Area Network) est un bus système série répandu dans l'industrie

SoftCore : processeur (CPU) implémenté sur un système reprogrammable comme un FPGA. On parle alors de système sur puce programmable (System on Programmable Chip ou SoPC).

Spacewire : réseau de communication, utilisé par l'ESA

Chipscope Pro : logiciel fournissant une solution de débogage sur FPGA

I. Présentation du problème

I.1. Contexte

Lors de ma 5ème année au sein de l'école Polytech'Lille, dans le département I.M.A. (Informatique Microélectronique et Automatique), pour mon projet de semestre S9, j'ai choisi de travailler sur une problématique liée aux systèmes temps réels.

Au cours de mon projet, j'ai travaillé sur des systèmes embarqués dans le domaine aérospatial, les systèmes qui tendent à augmenter leur puissance de calcul. Pour se faire, ils intègrent tout comme des ordinateurs commerciaux, des systèmes avec plusieurs processeurs.

Or dans ces systèmes temps réels, il arrive que certaines tâches soient considérées comme critiques. Afin d'illustrer le côté critique : prenons l'exemple d'un appui sur une pédale de frein d'une voiture. Il est nécessaire que le véhicule dans un temps borné. Si le freinage n'est pas réalisé à temps, l'utilisateur peut être mis en danger car le véhicule peut alors être hors de contrôle.

Ces systèmes intégrant maintenant plusieurs processeurs accèdent aux mêmes ressources (caches mémoire, contrôleur mémoire, réseau CAN, Ethernet, …).

Le système d'un point de vue électronique ne peut être capable à un temps donné, de satisfaire toutes les demandes d'accès en lecture et en écriture à la mémoire de plusieurs processeurs. On parle de « contention mémoire ». C'est-à-dire que le bus est saturé de transferts et que le système perd son côté temps réel.

Notre projet, lui s'est concentré sur un processeur développé par Gaisler/Aeroflex et celui-ci est utilisé dans l'aérospatial : le LEON3. Pour son développement, le processeur est écrit en VHDL et des cartes pour le développer ainsi que le tester existent.

C'est ainsi que nous avons choisi de faire un composant capable de suivre les accès mémoire d'une architecture comportant des processeurs LEON3.

I.2. Objectif du projet

Le sujet proposé était le suivant :

Des processeurs sont aujourd'hui testés par simulation sur FPGA avant d'être produits. Certains processeurs lors de leur audit de sécurité montrèrent des faiblesses sur l'accès à la mémoire et aux caches.

Pour répondre à cette problématique, le projet s'est décomposé en 2 parties :

- L'identification du problème et la localisation de la partie défaillante du processeur (repérage des lignes de code VHDL incriminées).
- La modification du code VHDL afin d'intégrer un nouveau composant innovant réalisant la sécurité de cette partie du processeur.

Un bon niveau de VHDL est nécessaire ainsi qu'une autonomie et une prédisposition (motivation) à la recherche de problèmes de sécurité sur systèmes.

I.3. Ressources utilisées

Cette partie a pour but de décrire les fondamentaux des ressources déjà existantes dans ce projet. Nous nous concentrons sur les détails dans les sous-parties concernant l'implémentation d'une solution.

I.3.a. Carte utilisée

Nous utilisons une carte de développement GR-PCI-XC5V qui dispose d'un FPGA de la famille Virtex5 de Xilinx sur celle-ci. Cette carte a été spécialement conçue pour le développement du LEON3 et son bus AMBA fonctionne à 50MHz. Nous utilisons le port JTAG pour la programmation du FPGA[1].

D'autres fonctionnalités sont disponibles sur cette carte :

- Le port PCI qui permet à la carte d'être mise dans une unité centrale d'ordinateur.
- Un contrôleur Ethernet Gigabit
- Une mémoire RAM de 256Mb (kvr133x64sc3l/256)

I.3.b. GRLIB de Gaisler

La bibliothèque GRLIB qui contient l'I.P.[2] du LEON3 contient aussi les ressources nécessaires pour faire un SoftCore. Elle est fournie par Aeroflex/Gaisler. L'infrastructure est disponible sous licence GNU GPL. Elle contient des I.P.s pour le processeur leon3, le PCI, l'USB, l'Ethernet et le contrôleur mémoire. Cette bibliothèque contient un grand nombre de composants connectés grâce au bus AMBA[3] comme montré sur la figure ci-dessous.

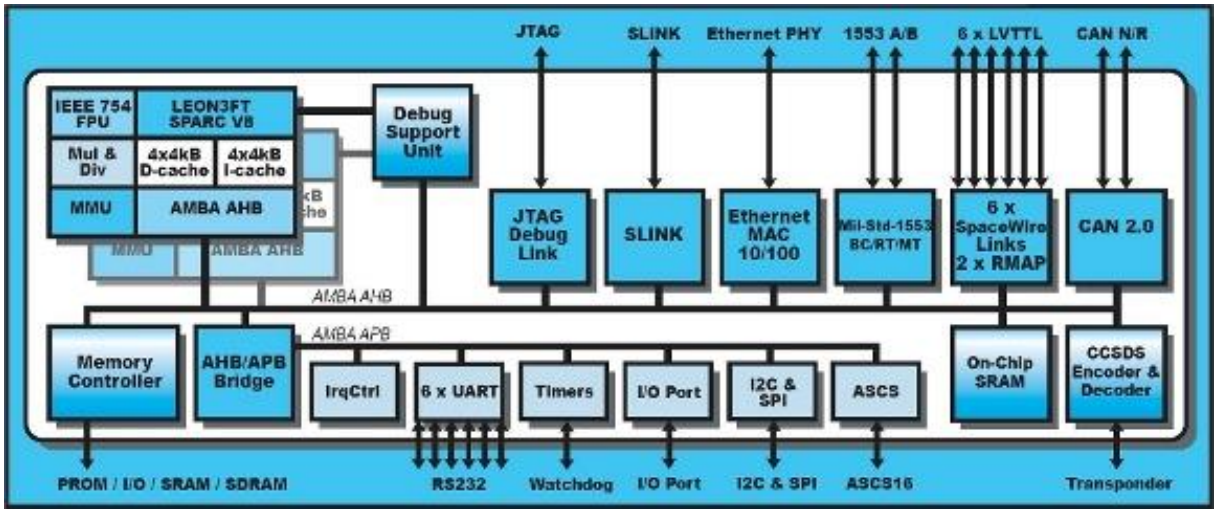


Figure 1 : Système LEON3 montrant le BUS AMBA AHB[4] et APB[5]

Le processeur LEON3 est un processeur 32-bit qui implémente toute la norme SPARC v8[6]. Il est possible de configurer l'architecture du design connecté au bus AMBA grâce aux outils fournis par Gaisler.

[1] Au travers de lignes de commandes
[2] Intellectual Properties
[3] http://www-micro.deis.unibo.it/~magagni/amba99.pdf
[4] Advanced High-performance Bus
[5] Advanced Peripheral Bus
[6] http://www.gaisler.com/doc/sparcv8.pdf

I.4. Cahier des charges du composant

Le composant :

- doit pouvoir être adressable par les processeurs à une adresse définie préalablement (en hardware)
- doit pouvoir être initialisé à certains états
    - l'accès en lecture ou en écriture par le processeur à l'adresse du composant doit être possible et donc il doit être adressable
- doit être réglable pour choisir le composant surveillé (en hardware)
- doit pouvoir faire la différence entre :
    - un accès en lecture ou écriture sur la zone surveillée
    - chaque maître qui accède à la zone surveillée

Nous avons étudié où placer le composant :

Position d'intégration	Plug & Play	Possibilité d'action sur la mémoire	Modifications		
			Arbitre de bus	Architecture du design	Contrôleur mém.
Dans l'arbitre de bus AHB	non	Indirectement	oui	oui	oui
Sur le bus AHB	oui	Indirectement	non	oui	oui
Dans le contrôleur mémoire	oui	Directement	non	non	oui

Tableau 1: comparatif des implantations possibles

Nous avons choisi d'implanter le composant dans le contrôleur mémoire[7].

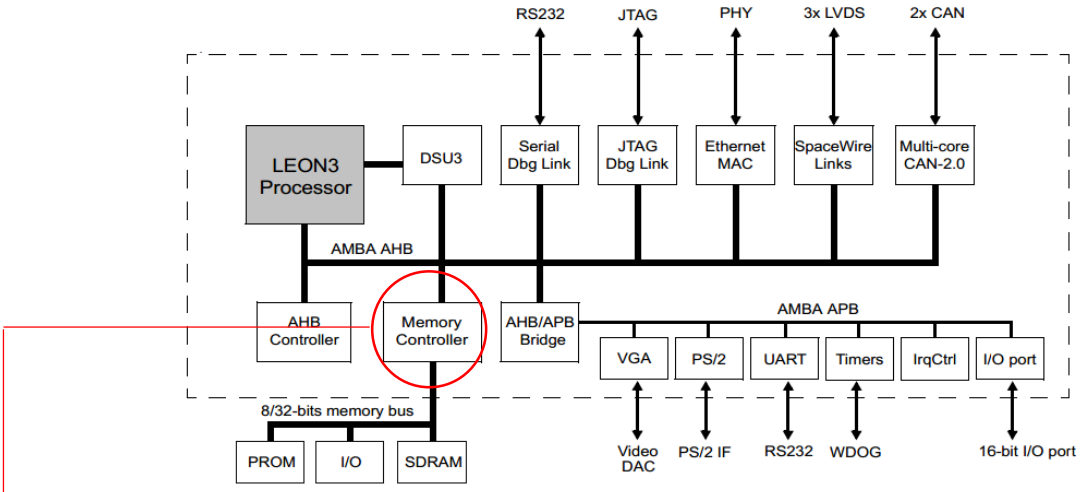
Les raisons de notre choix sont :

- La possibilité d'accéder aux signaux qui proviennent directement des mémoires.
- La réalisation d'un code moins sensible au niveau des ajouts/suppressions d'entrées/sorties.
- L'accès à la déclaration « plug & play » auprès du contrôleur de bus, ce qui le rendra accessible par les processeurs.

[7] L'implantation dans l'arbitre de bus a été envisagée, mais abandonnée, car jugée trop complexe.

I.4.a Position du composant dans le design

L'architecture abstraite proposée par Gaisler est la suivante :



Le composant reçoit les signaux des composants demandeurs d'informations : les processeurs (maîtres) sur le bus AHB et le DSU[8] (voir Implémentation II .4.a. Maîtres et esclaves) :

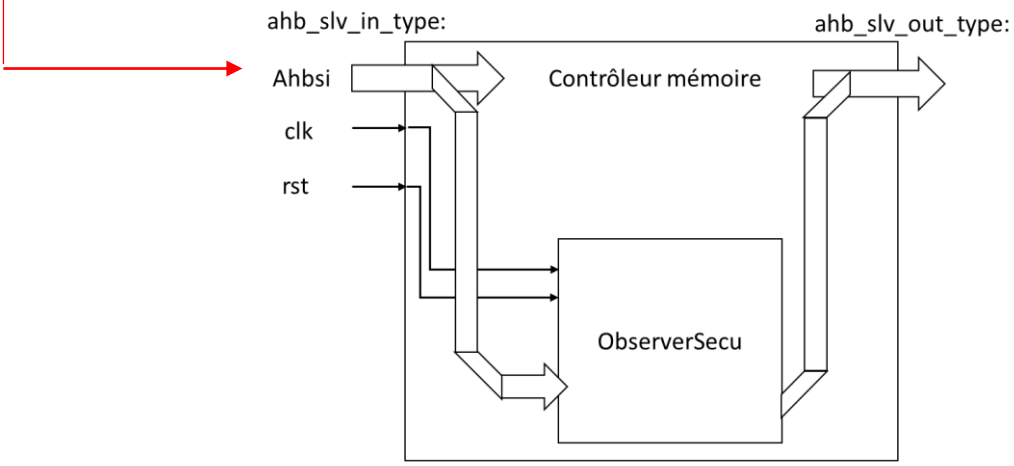


Figure 2 : Implantation dans le contrôleur mémoire

Le contrôleur mémoire est un esclave, d'après la GRLIB, il reçoit les signaux ahbsi de type « ahb_slv_in_type » (signal comportant l'ensemble des signaux d'entrées d'un esclave standard), renvoie le signal ahbso de type « ahb_slv_out_type » (signal comportant l'ensemble des signaux de sortie d'un esclave standard), avec clk qui est l'horloge et rst qui est la remise à zéro.

[8] Debug System Unit

I.5. Étapes du projet

Afin de mener à bien le projet, différents objectifs ont été posés :

- Etat de l'art de la documentation de la GRLIB et de la norme AMBA
- Prise en main de l'interface de configuration de l'architecture du processeur
- Réalisation d'un composant qui compte le nombre d'opérations pour une zone adressable spécifiée :
    - Prise en main de Grmon: les commandes « bas niveau » seront utiles pour réaliser des accès en écriture et en lecture sur un espace mémoire. (Cela va nous permettre de mettre en valeur le bon fonctionnement de notre composant : du point de vue du processeur)
    - Prise en main de ChipScope Pro : si un mauvais fonctionnement est constaté, il est nécessaire de débugger électroniquement le composant.
- Test du composant via Grmon
- Implémentation d'un cache L2 (voir II.4.a. Maîtres et esclaves) au sein du contrôleur mémoire

a. Etats de l'art de la documentation de la GRLIB et de la norme AMBA

La découverte de l'ensemble des commandes pour utiliser les outils fournis par Gaisler. L'appréciation de l'adaptation de la norme AMBA pour l'architecture du LEON3.

b. Prise en main de l'interface de configuration de l'architecture du processeur

Mise en place de l'ensemble de l'environnement de développement pour le projet ; les différentes variables d'environnement et utilitaires pour exécuter les scripts TCL/TK, réaliser ainsi une configuration de l'architecture qui prépare la partie software du projet (Grmon).

c. Réalisation d'un composant qui compte le nombre d'opérations pour une zone adressable spécifiée

La réalisation du VHDL qui permet d'implanter l'électronique qui réalise la fonction en hardware.

d. Test du composant via Grmon

Le composant sera testé via des commandes dans Grmon et corrigé dans le cas de défaut de fonctionnement de celui-ci.

e. Implémentation d'un cache L2 au sein du contrôleur mémoire

Enfin, si le temps le permet, un cache L2 peut être développé.

I.6. Gants prévisionnel

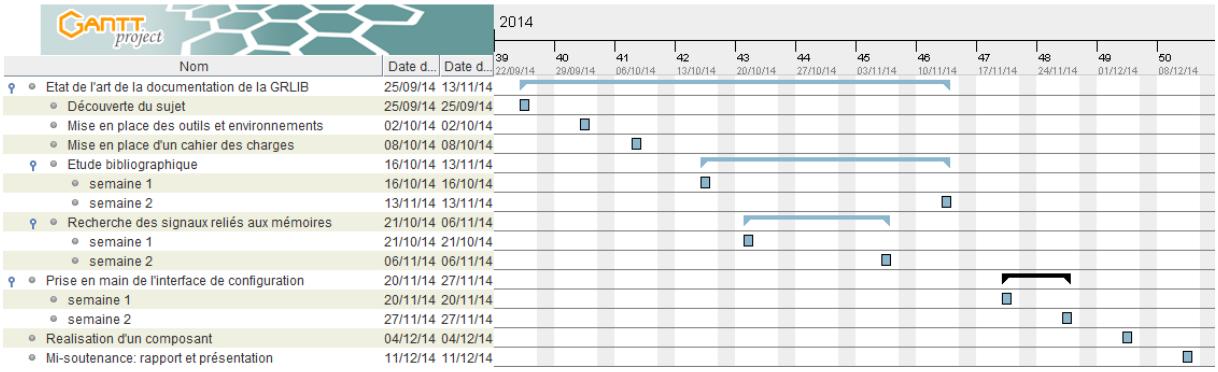


Figure 3 : Gant prévisionnel de septembre à décembre

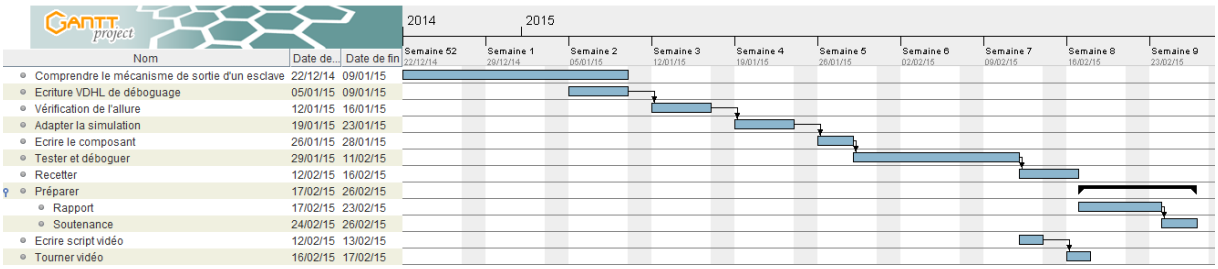


Figure 4 : Gant prévisionnel de décembre à février

II. Implémentation du composant et du cache L2

II.2.Bus AMBA

Le bus AMBA 2.0 est un bus intégré au design de la GRLIB. La norme AMBA est documentée, en accès libre et est utilisée pour les processeurs des System-on-chip (exemple : smartphones). La spécification contient trois types de BUS : AHB (Advance High-performance Bus), ASB (Advance System Bus) et APB (Advance Peripheral Bus). A savoir, seuls les bus AHB et APB sont utilisés dans la GRLIB.

II.3. Prise en main de l'interface de configuration de l'architecture avec la GRLIB

L'interface de réglage permet de choisir les fonctions à implémenter pour notre projet. Via l'interface configuration nous avons choisi :

- D'activer le JTAG « debug link[9] », indispensable pour la communication avec GRMON ; ce composant communiquera avec le DSU (Debug System Unit).
- De désactiver des périphériques à l'exception des réglages plus hauts :
    - On-chip AHB RAM, Ethernet, CAN, Spacewire, PCI, USB 2.0 Host Controller, USB 2.0 Device Controller, timers and I/O port.
        - Car ceux-ci sont inutiles et même indisponibles pour certains.

II.4. AHB

II.4.a. Maitres et Esclaves

L'arbitre/contrôleur de bus est configurable afin de faire différents modes d'arbitrage, en « round-robin » ou en mode « priority-fix ».Quand une requête d'un maître est prise en charge par l'arbitre de BUS, la requête est transmisse à l'ensemble des esclaves et l'arbitre sélectionne l'esclave concerné. Le bus AHB est multiplexé et donc peut être implémenté dans des FPGAs, comme on peut d'ailleurs le voir sur la figure suivante :

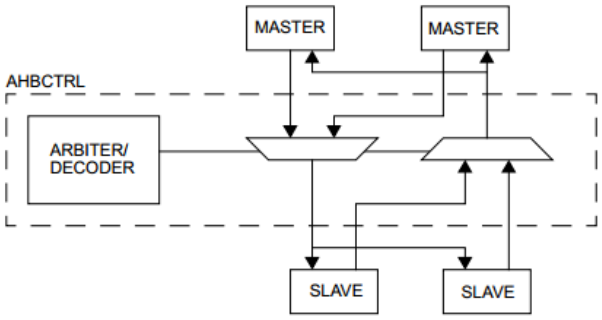


Figure 5 : Schéma de principe du contrôleur AHB (AHBCTRL)

L'arbitre de bus autorise un seul maître à la fois à communiquer aux esclaves.

Les processeurs étant maîtres et le contrôleur mémoire esclave sont sur le bus AHB, notre composant devra surveiller les actions des composants (maîtres AHB) qui accèdent au contrôleur mémoire.

[9] Lien de débogage

II.4.b. PLUG & PLAY de l'arbitre de bus

L'arbitre de bus de la GRLIB fournit des fonctionnalités de déclaration de plan mémoire. Les esclaves déclarent via un signal « hconfig » leurs adresses mémoires en fournissant un masque et une adresse. Cela permet ensuite de déterminer au contrôleur de bus AHB, l'espace mémoire occupé et ainsi de sélectionner les composants ciblés.

II.4.b.1 Décodage mémoire de l'arbitre de bus

L'arbitre de bus fournit une fonction de décodage mémoire, voir la figure suivante extrait de la norme. Celle-ci est configurée grâce à la fonction « plug & play » et sélectionne l'esclave correspondant à l'adresse d'opération sur le bus AMBA.

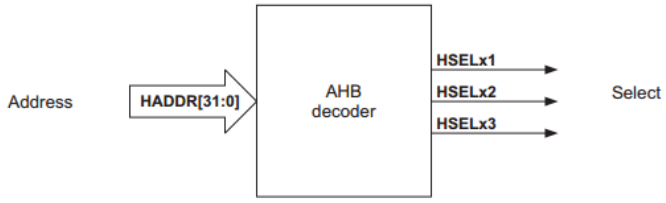


Figure 3-35 AHB decoder interface diagram

Figure 6 : Interface de décodage selon la norme AMBA 2.0

On utilise la fonction « plug & play » et on déclare notre composant afin de changer la cartographie mémoire :

Sans		Avec	
Entité	Adresse	Entité	Adresse
ROM	0x000 00000 0x1FF FFFFFF	ROM	0x000 00000 0x1FF FFFFFF
I/O	0x200 00000 0x3FF FFFFFF	I/O	0x200 00000 0x3FF FFFFFF
SDRAM/ RAM	0x400 00000 0x7FF FFFFFF	SDRAM/ RAM	0x400 00000 0x7FF FFFFFF
		ObserverS ecu	0xA00 00000 0xA00 FFFFFF

Tableau 2 : Plan mémoire avant et après l'ajout du composant

Notre composant est visible dans GRMON donc adressable ; voir en annexe « Reconnaissance du composant dans GRMON ».

II .4.c. Transfert AHB et allure des signaux d'accès d'un maître sur le contrôleur mémoire

Le composant est connecté et reconnu par le contrôleur de bus AMBA. La communication peut être alors établie entre un maître et notre composant.

La communication et le transfert des données se fait par des signaux définits par la norme AMBA. Voici ci-dessous, une allure de ces signaux. Lors d'un transfert de données, les signaux ont cette allure d'après la norme AMBA :

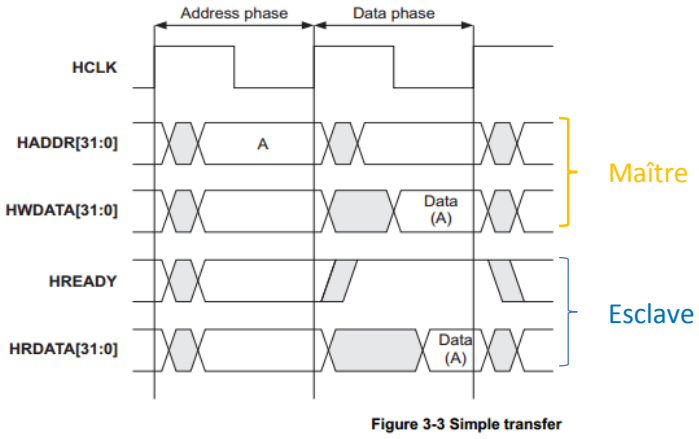


Figure 7 : Simple transfert d'après la norme AMBA

Descriptifs des signaux :

Signal	Qualité	Commentaire
HCLK	Horloge du bus / Commune à tous les composants	
HADDR (32 bits)	Sortie d'adressage d'un maître sur le bus / Entrée d'adressage des esclaves du bus	Conséquence directe du multiplexage du bus AHB
HWDATA (32 bits)	Sortie de donnée d'un maître sur le bus / Entrée de données des esclaves du bus	Conséquence directe du multiplexage du bus AHB
HRDATA (32 bits)	Sortie de l'esclave	Sortie de données de l'esclave
HREADY	Sortie de l'esclave	Signal mis à un à la fin d'un transfert

Tableau 3 : description des signaux de la norme AMBA

Dans notre design en simulation, les signaux sont similaires :

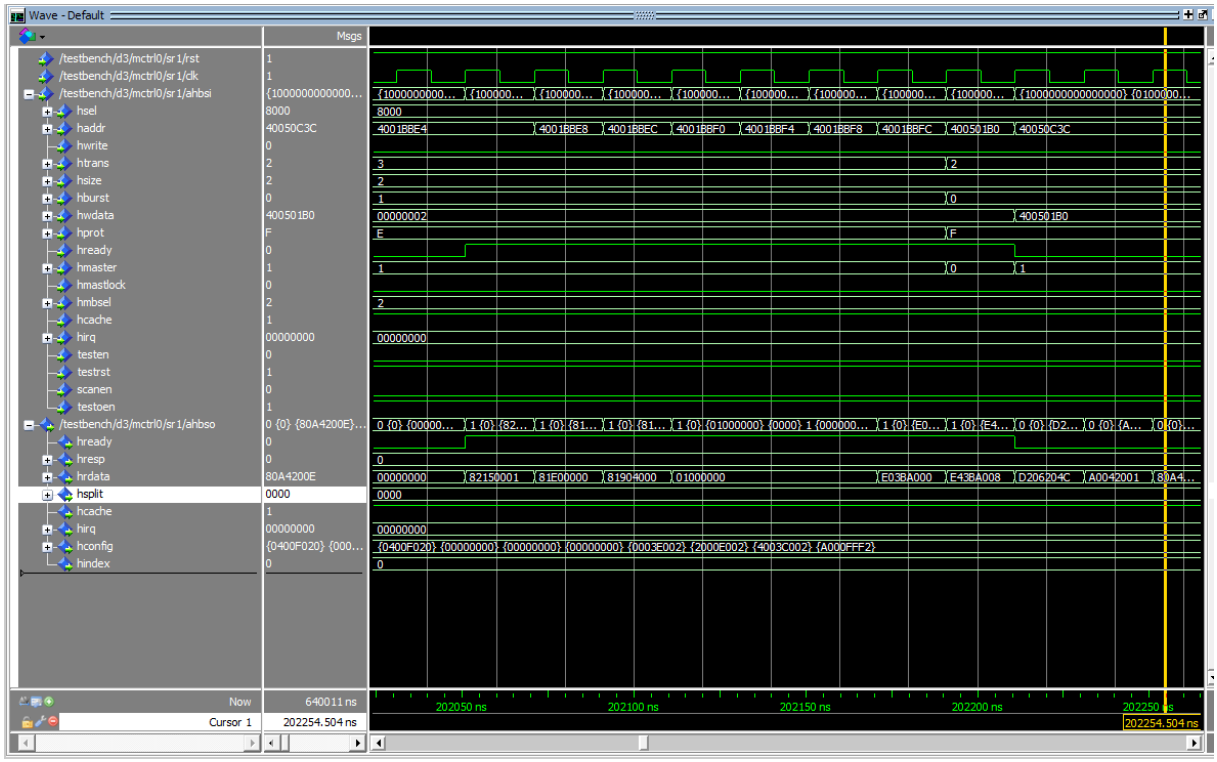
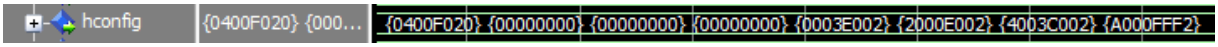


Figure 8 : Extrait de la simulation du projet entier

La déclaration de notre composant {A00 0 FFF 2} est visible dans le « hconfig » du contrôleur mémoire dans la figure ci-dessus (avant-dernière ligne) :



La console dans la simulation affiche une entrée correspondante :

```
# ahbctrl : slv0 : European Space Agency   Leon2 Memory Controller
# ahbctrl :      memory at 0x00000000, size 512 Mbyte, cacheable, prefetch
# ahbctrl :      memory at 0x20000000, size 512 Mbyte
# ahbctrl :      memory at 0x40000000, size 1024 Mbyte, cacheable, prefetch
# ahbctrl :      memory at 0xa0000000, size 1 Mbyte
```

Figure 9: Extrait des traces de simulation dans la console de ModelSim PE

II.5. Réalisation du composant

Le banc de test équivalent du composant isolé est le suivant :

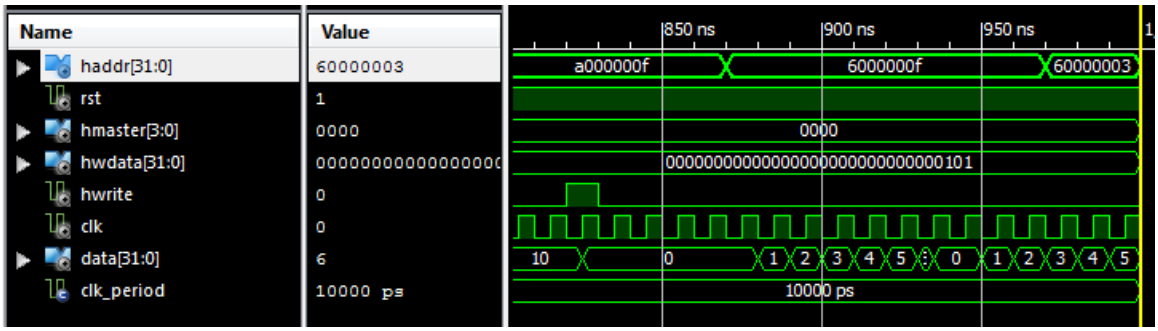


Figure 10 : Composant simulé seul

Le signal data est connecté à hrdata, le composant fournit en permanence son résultat au contrôleur mémoire et n'a pas de cache, ni de « prefetch ». Une écriture permet de configurer sa valeur maximale de comptage.

II.5.a. Schéma du composant

Schéma proposé :

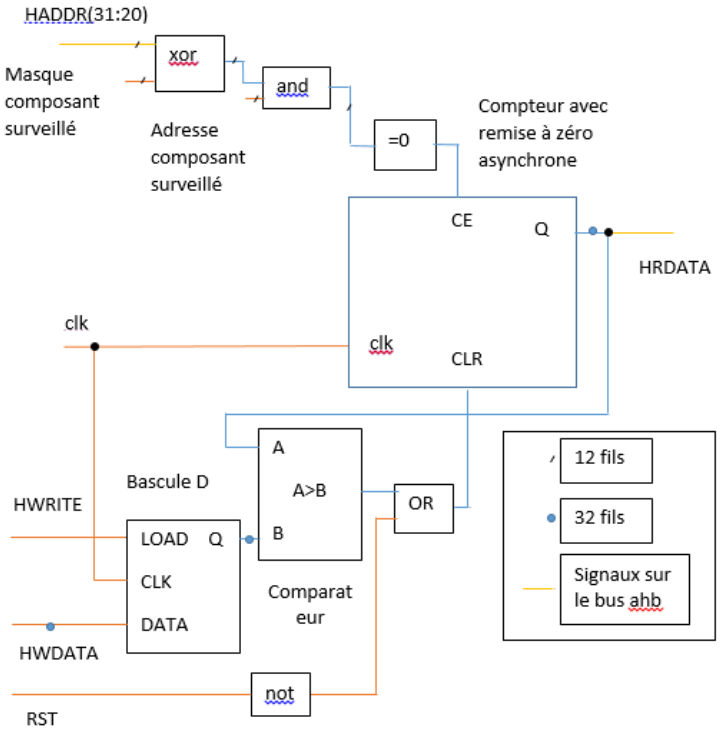


Figure 11 : Schéma de principe du composant

L'activation du comptage de notre composant se fait de la même manière que la sélection des composants par l'arbitre de bus. Il nous faut régler l'adresse et le masque du composant surveillé et le signal de sélection ainsi reconstitué permet d'activer le comptage.Le signal RST est la remise à zéro (actif à l'état bas) et le signal HWRITE indique s'il y a écriture ou non.

La fonction d'écriture de la valeur maximale est en bas de la figure précédente. Elle est faite grâce au comparateur et la bascule D.

II.5.b. Test et correction

Lecture

Une lecture dans GRMON renvoie la valeur du compteur ; voir en annexe « Lecture du composant ».

Écriture

Une Écriture dans GRMON à l'adresse du composant n'a pas d'effet sur le composant, Chip Scope doit fonctionner pour permettre une correction plus aisée du composant et ainsi voir les signaux réels.

Chip Scope fournit la possibilité d'implanter des analyseurs logiques dans un design VHDL[10]. Chip Scope collecte les états des signaux dans une petite mémoire jusqu'à que celle-ci soit pleine. Le contenu est envoyé via la liaison JTAG. Pour être ensuite visualisé sur une interface graphique sur un ordinateur.

Correction du composant

Le composant n'est pas corrigé, car un retour des signaux réels via Chip Scope pro est obligatoire. En effet, Une simulation VHDL ne reflète pas toute la réalité matérielle dans lequel le composant est implanté.

L'insertion des analyseurs logiques de Chip Scope crée des erreurs multiples qui nous empêchent de passer sur la carte pour ensuite récupérer les signaux réels.

[10] Après la synthétisation

II.6. Réflexion sur un cache L2

Nous avons donc décidé de travailler sur un cache L2.

Afin de limiter le problème de contention mémoire, nous avons commencé un cache L2.

II.6.a Etats de l'art : cache L1

D'un point vu électronique, l'architecture comporte un cache L1 (petit et rapide), et le contrôleur mémoire (plus gros et plus lent). Le principe est le suivant :

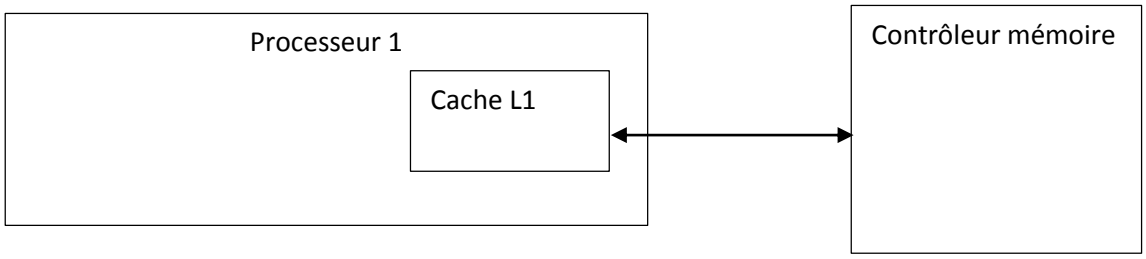


Figure 12 : Schéma abstrait d'un processeur communiquant avec le contrôleur mémoire

Le processus d'accès mémoire se déroule de la manière suivante :

1) Un processeur demande une information
2) Le cache L1 vérifie s'il est en possession de l'information :
    a. Si oui, il fournit l'information au processeur (cache hit)
    b. Si non, il demande au contrôleur mémoire (cache miss)
3) Le contrôleur mémoire possède toutes les informations et donc renvoie l'information au cache L1.

Les mémoires caches ont été faites pour accélérer les systèmes informatiques en se basant sur deux principes : le principe de localité spatiale[11] et le principe de localité temporelle[12]. Cependant, si des tâches font suffisamment d'accès mémoire pour renouveler l'intégralité du contenu du cache ; on perd le gain acquis grâce à ceux-ci. Cela a pour résultat une augmentation du temps de changement de contexte.

Par exemple : prenons un accès à une matrice de grande taille pour réaliser un traitement d'image. Le chronogramme suivant montre l'influence de T2 qui est la tâche qui renouvèle l'intégralité du cache.

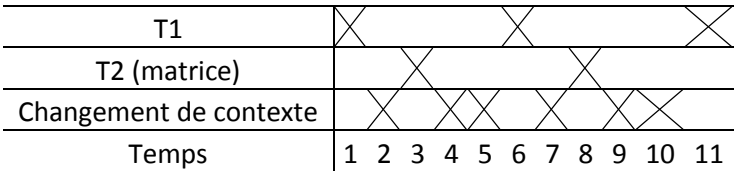


Figure 13 : Chronogramme de deux tâches temps réels

[11] L'accès à une donnée X va probablement être suivant d'un accès à une zone proche de X

[12] L'accès à un instant donné à de fortes chances de se reproduire dans la suite immédiate du programme

II.6.b. Cache mémoire L2

Un cache L2 se dispose de manière abstraite suivante :

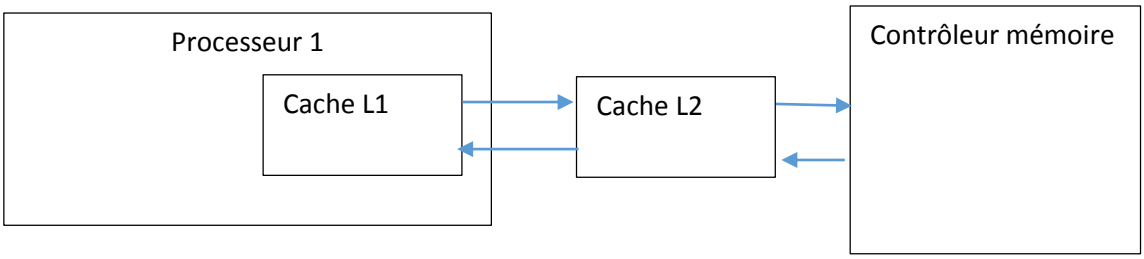


Figure 14 : Schéma abstrait d'un processeur communiquant avec le contrôleur mémoire au travers du cache L2

Le processus d'accès mémoire se déroule de la manière suivante :

1) Un processeur demande une information
2) Le cache L1 vérifie s'il est en possession de l'information :
    a. Si oui, il fournit l'information au processeur (cache hit)
    b. Si non, il demande au cache L2 (cache miss)
3) Le cache L2 vérifie s'il est en possession de l'information :
    a. Si oui, il fournit l'information au cache L1 (cache hit)
    b. Si non, il demande au contrôleur mémoire (cache miss)
4) Le contrôleur mémoire possède toutes les informations et donc renvoie l'information au processeur au travers des caches L2 puis L1.

Le but de celui-ci dans notre cas est de fournir une information provenant d'une mémoire ayant besoin de « prefetch » pour obtenir une réponse.

II.6.c. Implantation et comportement

On l'implante dans le contrôleur mémoire, le composant vérifie s'il a déjà eu cette requête. Si oui, il répond et signale au contrôleur de bus la fin du transfert via le signal approprié (HREADY).

Si non, il laisse le contrôleur mémoire répondre et stocke l'information pour répondre à la place du contrôleur. (Dans le cas où l'information est demandée dans les quatre prochaines requêtes.)

II.6.d Schéma du cache

Schéma proposé :

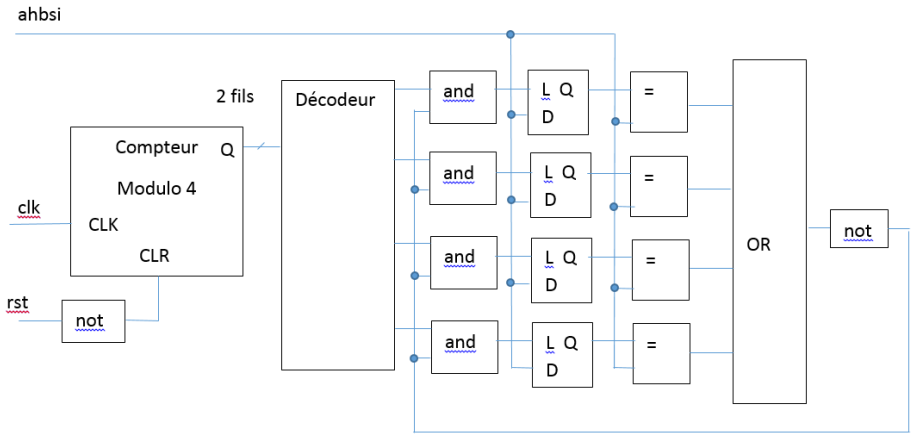


Figure 15 : Proposition de schéma du cache pour l'entrée

Le signal « ahbsi » est l'entrée du bus ahb du contrôleur mémoire. L'inconvénient de ce schéma est qu'il est sensible au changement de manière asynchrone, mais nous n'avons pas d'autre proposition à faire pour le moment.

Le but de notre composant est de faire une correspondance entre les signaux d'entrée du contrôleur et la sortie correspondante.

Un code correspondant a été développé et testé, mais une correction doit être faite.

III. Ressenti sur le projet

III.1. Difficultés rencontrées

J'ai rencontré deux principaux problèmes sur ce projet :

- La mise en place de l'environnement de développement
- La manipulation de l'outil ISE Xilinx

La mise en place de l'environnement de développement n'a pas été facile au départ. En effet, les outils fournis par Gaisler fonctionnaient, mais nécessitent bien d'autres outils périphériques.

On peut parler notamment des bibliothèques TCL/TK qui nous ont posé problème, car il fallait une version bien précise de celle-ci. La première solution retenue fut de récupérer les fichiers sources de la bibliothèque et de la compiler. Cette solution n'a pas été fructueuse, du fait que compilation ne fonctionnait pas. Suite à des recherches sur internet, nous avons pu trouver une version compilée qui correspondait et qui nous as permis de continuer.

L'autre problème rencontré est que l'outil (ISE Xilinx) nous a souvent supprimé notre composant durant le processus d'optimisation. La plupart du temps nos signaux et le composant étaient absents. Nous avons donc appris à utiliser l'attribut « keep » qui permet de garder un signal.

Quant à l'environnement (ISE Xilinx) et à la synthétisation, dans le cas où ils n'arrivent pas à identifier un composant ; il considère qu'il doit être une brique de base (ou une boite noire, « black box »). Et ceci sans aucune vérification (lors de l'étape « Synthetise – XST »).Comme cette vérification est seulement faite au moment de l'étape « Translate » de la partie « Implementation ».

Ceci pourrait paraître bénin, mais ceci est en fait crucial. En effet, on accède à la simulation, mais on ne peut générer de « bitstream » pour tester le design sur la carte. Ce qui fait que l'on a épluché les 878ko de la console de l'étape « Synthesize – XST » pour trouver l'origine du problème. Les erreurs se sont avérées provenir de plusieurs composants, soit mal incluses, soit inexistantes. Voir en annexe « ISE Xilinx workflow ».

III.2. Bilan personnel

J'ai trouvé ce projet très enrichissant. Tout d'abord d'un point de vue technique, il m'a permis de mettre à profit des compétences acquises durant la formation et de découvrir de nouveaux langages et environnements.

J'ai pu apprendre à faire du VHDL avancé ainsi que voir l'aspect software et hardware d'un SoC (System on Chip). Le niveau de complexité du code rencontré est notable, mais je trouve cela valorisant d'avoir pu travailler sur cette problématique.

Deux points sur quatre du cahier des charges sont remplis : le composant est adressable et il est possible de régler l'espace mémoire surveillé. Les deux autres (différentiation des maîtres et distinction entre lectures ou écritures) sont quant à eux non réalisés.

En effet, nous ne sommes pas arrivés à confirmer de manière formelle sur la carte, le bon fonctionnement des deux premiers points réalisés. Il serait intéressant à l'avenir de pouvoir confirmer ce fonctionnement et mettre en place les deux dernières fonctionnalités.

IV. Diagramme de gant final

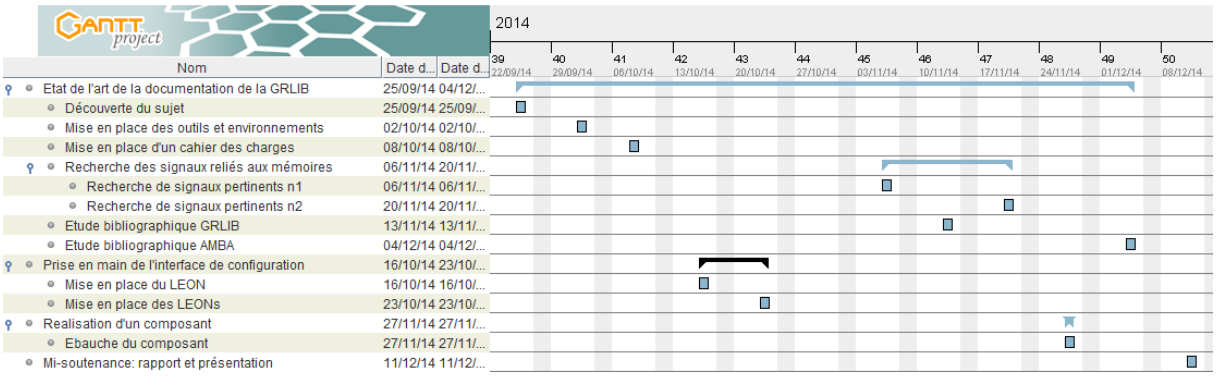


Figure 16 : Gant réel de septembre à décembre

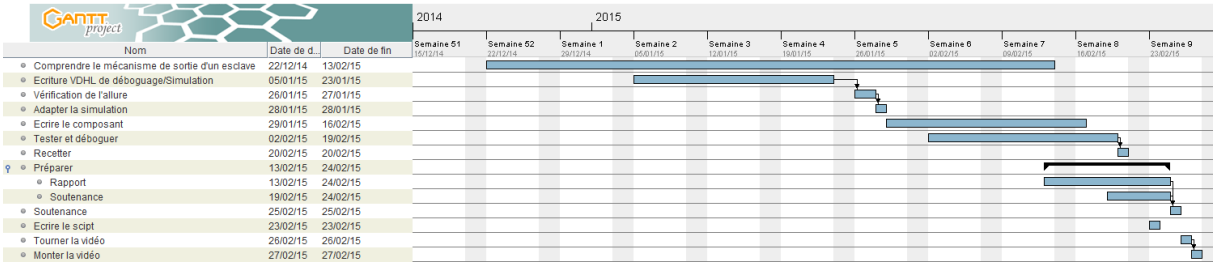


Figure 17: Gant réel de décembre à février

V. Conclusion

J'ai eu la chance de pouvoir travailler sur ce projet de fin d'études lors de ma cinquième année dans la spécialité Informatique Microélectronique et Automatique à Polytech'Lille.

Le projet consistait à créer un composant servant à faire des bancs de test pour l'utilisation mémoire. Le composant a été implanté dans le contrôleur et est capable de répondre à une sollicitation en lecture.

J'ai pu lors de ce projet appréhender les problèmes inhérents de la recherche et avoir une pensée constructive malgré un problème complexe. Cette expérience me sera utile dans la gestion de projet et m'a permis de gagner en efficacité lors de la rencontre de nouvelles problématiques.

Annexes

Reconnaissance du composant

Notre composant est adressable pour les composants sur le bus AHB, voici un extrait du résultat de la commande « info sys » dans GRMON :

```
grmon2> info sys
  […]
  mctrl0    European Space Agency  LEON2 Memory Controller
        AHB: 00000000 – 20000000
        AHB: 20000000 – 40000000
        AHB: 40000000 – 80000000
        AHB: A0000000 – A0100000
        APB: 80000000 – 80000100
        32-bit prom @ 0x00000000
        32-bit static ram: 1 * 8192 kbyte @ 0x40000000
        64-bit sdram: 2 * 128 Mbyte @ 0x60000000
        col 9, cas 2, ref 7.8 us
[…]
```

Annexe 1 : En gras, on voit la réponse de notre composant

La commande renvoie l'intervalle d'adressage de la manière suivante :

Adresse de début – adresse de fin+1.

Lecture du compteur

La commande « mem 0xA0000000 » dans Grmon permet de lire le contenu de la mémoire à l'adresse 0xa0000000 sur le bus ahb. Voici le résultat :

```
grmon2> mem 0xa0000000
  0xa0000000  41d05d71  41d05d71  41d05d71  41d05d71   A.]qA.]qA.]qA.]q
  0xa0000010  41d05d71  41d05d71  41d05d71  41d05d71   A.]qA.]qA.]qA.]q
  0xa0000020  41d05d71  41d05d71  41d05d71  41d05d71   A.]qA.]qA.]qA.]q
  0xa0000030  41d05d71  41d05d71  41d05d71  41d05d71   A.]qA.]qA.]qA.]q

grmon2> mem 0xa0000000
  0xa0000000  41d09011  41d09011  41d09011  41d09011   A...A...A...A...
  0xa0000010  41d09011  41d09011  41d09011  41d09011   A...A...A...A...
  0xa0000020  41d09011  41d09011  41d09011  41d09011   A...A...A...A...
  0xa0000030  41d09011  41d09011  41d09011  41d09011   A...A...A...A...
```

Annexe 2 : En gras, on voit la valeur du compteur dans notre composant

Notre composant répond toujours le même résultat, comme cela était prévu.

ISE Xilinx workflow

Les étapes de génération d'un bitstream pour un FPGA sont sur la figure ci-dessous :

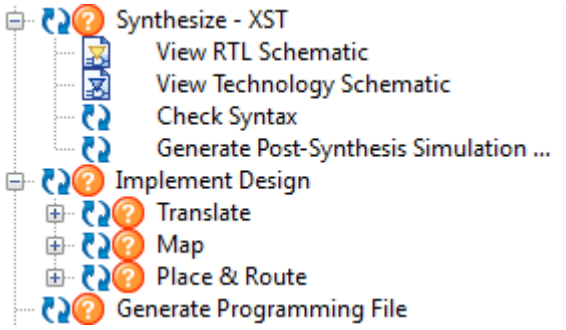


Figure 18 : Etapes de génération de "bitstream"

Ainsi la première étape « Synthetize – XST » que l'on peut voir représenté sur la figure ci-dessous :

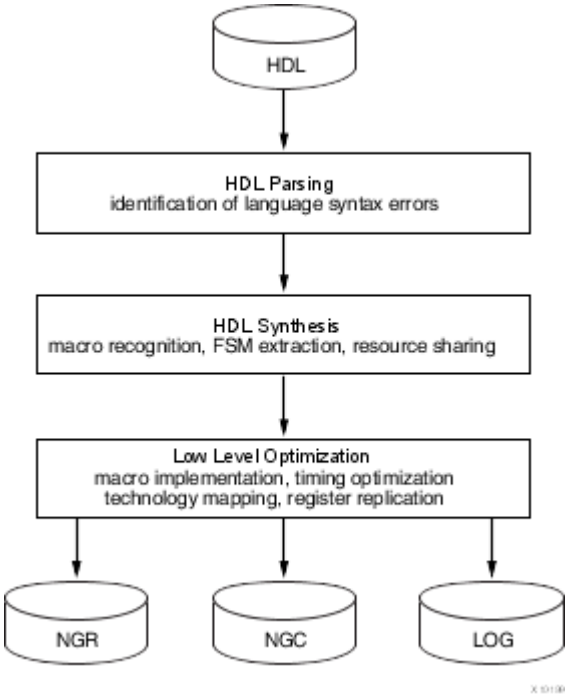


Figure 19 : Procedure de synthése

Après une synthése, les fichiers .ngr, .ngc et .log sont générer. Remarquons que le ngc peut ne pas être générer en utilisant les outils de Gaisler. Cela provoque une erreur pendant la prochaine étape l'implémentation.

Ensuite il y a l'implémentation (implementation) fait les étapes suivantes:

Translate - fusionne les signaux identiques et inscrits dans un fichier de design Xilinx et qui réalise la reconnaissance de block de base

Map - Loge le design dans les ressources disponibles pour la puce cible

Place and Route - Place et relie le design en fonction des contraintes de temps

Generate Programming File - Créer un bitstream qui peut être chargé dans la puce